REMARKS

I. INTRODUCTION

Claims 1-60 remain pending in the present application. Claims 36 and 37 have been amended. No new matter has been added. In view of the following remarks, it is respectfully submitted that all of the presently pending claims are allowable.

II. THE 35 U.S.C. § 101 REJECTIONS SHOULD BE WITHDRAWN

Claims 36 and 37 stand rejected under the 35 U.S.C. § 101, the Examiner contending that the claimed invention is directed to non-statutory subject matter. The Applicant has amended the claims to produce a useful result when fixed in a tangible medium. Thus, it is respectfully submitted that the rejections be withdrawn.

III. THE 35 U.S.C. § 102(b) REJECTIONS SHOULD BE WITHDRAWN

Claims 1-41 and 43-60 stand rejected under 35 U.S.C. § 102(b) as being anticipated by John Levine, "Linker and Loaders, chapter 6," June 1999 [online] accessed 08/15/2005, retrieved from Internet <URL: http://www.iecc.com/linker/linker06.txt>, 9 pages (hereinafter "Levine").

Levine generally describes a technique used by an archive format for creating libraries. A library is handled by a program linker to resolve symbol references within a program. (*See* Levine, "Library formats," p. 1). The libraries may contain selected object files, or routines, that resolve undefined symbols, and these object files may be used by linkers and

loaders in order to automate symbol resolution. (*See* Id.). In addition, these collections of object files within the libraries may be created using various formats, wherein the simplest format for a library is a sequence of object modules. (*See* Id.). The format of the linker libraries implemented in UNIX and Windows is an "archive" format, which may be used for collections of any types of files. (*See* Id.).

For the creation of libraries, the archive formats may use a variety of techniques depending on the support provided by a given operating system. (*See* Id. at "Creating libraries," p. 4). In order to deal with the issue of ordering the object files within a created archive library, UNIX systems contains two programs, "lorder" and "tsort," to help in the creation process. (*See* Id.). Lorder is used to produce a dependency list of the object files that reference specific symbols in other object files. (*See* Id.). This is accomplished by extracting the symbols using a symbol listing utility, text processing the symbols, and using standard sort and join utilities to create an output. (*See* Id.). Tsort performs a topological sort on the output in order to produce a sorted list of object files. (*See* Id.). Thus, each symbol may be defined after all the references to it, thereby allowing all undefined references to be resolved over a single sequential pass. (*See* Id.). Therefore, an archive library may be created with the lorder and tsort programs where the output of lorder is used to control the archive library. (*See* Id.).

The present invention relates to a new approach to the reordering software modules for improving the linking process, particularly on systems in which the code does not fit entirely in the system memory of the computer performing the linking operation. When a software module is compiled, loaded, and linked according the present invention, the compiled software module may include "backward references," or pointers and indices that reference an

earlier portion of that compiled software module. (*See* Specification, p. 5, ll. 15-24).

Claim 1 recites "receiving a software module, the software module including references to location within the software module, at least some of the references being backward references." The Examiner contends that this limitation is anticipated by Levine's discussion of the use of the lorder and tsort programs to arrange object files within an archive library in proper dependency order to allow a single sequential linker pass to resolve all undefined references. (*See* Office Action, pp. 3-4, ¶ 8).

The Examiner appears to be equating the receiving of a software module having backwards references of the present invention to the creation of an archive library of Levine. As described above and as clarified in an exemplary embodiment of the present invention, "backward references" are references to a location within a software module and are included in the received software module. (*See* Specification, p. 5, ll. 15-24). Two types of backwards references may be included in a software module reordered according to the present invention, i.e., the example reorder software module. (*See* Id. at p. 10, l. 30 to p. 11, l. 32). The two backwards references within the received software module (first to a symbol string table and second to an index in a symbol table) may be used for linking information between a section header table and the symbol table within the example reorder software module. (*See* Id. at p. 12, ll. 7-27). Furthermore, a linker may be built with the information contained within the backward references that are used in the example reordered software module. (*See* Id. at p. 11, ll. 32-34). Thus, the backward references are references defined or found within the received software module, such as data sections and instruction sections. In other words, having a software module

that contains backward references to the section header and to the symbol table will allow for a linker to use the references to avoid the need to transition back-and-forth, non-sequentially while reading the software module.

In contrast, Levine describes the creation of libraries using the "ar" command in UNIX, which combines object files into archives to create a symbol directory. (*See* Id. at "Creating libraries," p. 4). Levine further describes the use of the lorder and tsort programs for the creation of libraries for a format that does not have a symbol directory. (*See* Id.). Specifically, the lorder program generates a dependency list of symbol references, while the tsort program generates a sorted dependency list that defines each symbol after its references. (*See* Id.). The generation of a library directory from a sorted list of symbol references by the lorder and tsort programs does not serve the an equivalent function as receiving a software module that includes pointers or indices that reference an earlier portion of the software module. The process of Levine is merely a method for creating an archive library for object files that lacks a symbol directory, or have undefined symbols. Levine does not teach or disclose a software module that includes reference to information from an earlier portion of the software module to allow for a linker to be built from the referenced information for efficient processing. Thus, it is respectfully submitted that Levine's disclosure neither teaches not suggests "receiving a software module, the software module including references to location within the software module, at least some of the references being backward references" as recited in claim 1.

Furthermore, claim 1 goes on to recites "reordering components of the software module to remove at least some of the backward references." In other words, by placing the components of the software module in a predetermined order, many of the backward references

within the software module may be remove when processed by the linker. The Examiner appears to equate the reordering element of the present invention to the symbol resolution described in the Levine's use of the lorder and tsort programs. However, the process of symbol resolution is not the removing a backward reference of a software module. As described by Levine, the sorted dependency list of symbol references is used for an efficient resolution of all undefined references. (*See* Levine at "Creating libraries," p. 4). The resolution of symbols is the process of associating each symbol reference with a single definition. The process of Levine fails to disclose any step equivalent to the removal of a backward reference or the removal of any pointers or indices within a software module that reference an earlier portion of the software module. Thus, it is respectfully submitted that Levine's disclosure neither teaches not suggests "reordering components of the software module to remove at least some of the backward references" as recited in claim 1.

Applicant respectfully submits that for at least the reasons stated above, claim 1 of the present application is not anticipated by Levine, and request that the rejection of this claim be withdrawn. As claims 2-8, and 40-54 depend from, and therefore include all of the limitations of claim 1, it is respectfully submitted that these claims are also allowable.

The Examiner rejected claim 9 as being anticipated by the discussion of creating libraries in Levine. (See Office Action, p. 5, ¶ 8). Claim 9 has similar recitations to claim 1. In particular, claim 9 recites "a software module including references to locations within the software module, at least some of the references being backward references, the reorder module configured to reorder components of the software module and remove at least some of the backward references." Thus, Applicant respectfully submits that for at least the reasons

discussed above with respect to claim 1, claim 9 should also be allowed. Therefore, Applicant requests that the rejection of claim 9, and the rejections of claims 10-15, which depend from and include all the limitations of claim 9, be withdrawn.

The Examiner rejected claim 16 as being anticipated by the discussion of creating libraries in Levine. (See Office Action, p. 6, ¶ 8). Claim 16 recites "receiving a software module sequentially, the software module having at least one symbol reference; linking the software module onto a target memory space; and resolving the at least one symbol reference without storing the entire software module in local memory while the symbol reference is resolved." As described above and as described by Levine, the lorder and tsort programs perform functions on object files to define symbols and create an archive library. The process of Levine fails to teach or disclose the method of linking the software module onto a target memory space or using symbol resolution without storing the entire software module in local memory. In fact, Levine fails to suggest any method as to the saving of an amount of the software module. Specifically, lorder produces a dependency list of symbol references and tsort produces a sorted dependency list of symbol references. Levine does not describe the saving of the software module, the amount of the software module saved, nor the location of where the software module is saved. Thus, it is respectfully submitted that Levine's disclosure neither teaches not suggests "receiving a software module sequentially, the software module having at least one symbol reference; linking the software module onto a target memory space; and resolving the at least one symbol reference without storing the entire software module in local memory while the symbol reference is resolved"as recited in claim 16.

Applicant respectfully submits that for at least the reasons discussed above, claim

16 of the present invention is not anticipated by Levine, and request that the rejection of this claim be withdrawn. Therefore, Applicant requests that the rejection of claim 16, and the rejections of claims 17-22, which depend from and include all the limitations of claim 16, be withdrawn.

The Examiner rejected claim 23 as being anticipated by the discussion of creating libraries in Levine. (See Office Action, p. 6-7, ¶ 8). Claim 23 has similar recitations to claim 16. In particular, claim 23 recites "a the linker configured to resolve the symbol reference, the linker configured to store less than the entire software module in local memory during the resolution of the at least one symbol reference." Thus, Applicant respectfully submits that for at least the reasons discussed above with respect to claim 16, claim 23 should also be allowed. Therefore, Applicant requests that the rejection of claim 23, and the rejections of claims 24-35, which depend from and include all the limitations of claim 9, be withdrawn.

The Examiner rejected claim 36 as being anticipated by the discussion of creating libraries in Levine. (See Office Action, p. 7-8, ¶ 8). Claim 36 has been amended to disclose similar recitations to claim 16. In particular, claim 36 recites "receiving a software module sequentially, the software module having at least one symbol reference; linking the software module onto a target memory space; and resolving the at least one symbol reference without storing the entire software module in local memory while the symbol reference is resolved." Thus, Applicant respectfully submits that for at least the reasons discussed above with respect to claim 16, claim 36 should also be allowed. Therefore, Applicant requests that the rejection of claim 36, and the rejection of claim 37, which depend from and include all the limitations of claim 36, be withdrawn.

The Examiner rejected claim 38 as being anticipated by the discussion of creating libraries in Levine. (See Office Action, p. 8, ¶ 8). Claim 38 has similar recitations to claim 1. In particular, claim 9 recites "receiving a software module including references to locations within the software module, at least some of the references being backward references, the reorder module configured to reorder components of the software module and remove at least some of the backward references." Thus, Applicant respectfully submits that for at least the reasons discussed above with respect to claim 1, claim 38 should also be allowed.

The Examiner rejected claim 39 as being anticipated by the discussion of creating libraries in Levine. (See Office Action, p. 8-9, ¶ 8). Claim 39 has similar recitations to claim 16. In particular, claim 39 recites "receiving a software module sequentially, the software module having at least one symbol reference; linking the software module onto a target memory space; and resolving the at least one symbol reference without storing the entire software module in local memory while the symbol reference is resolved." Thus, Applicant respectfully submits that for at least the reasons discussed above with respect to claim 16, claim 39 should also be allowed.

The Examiner rejected claim 55 as being anticipated by the discussion of creating libraries in Levine. (See Office Action, p. 8-9, ¶ 8). Claim 55 has similar recitations to claim 1. In particular, claim 55 recites "the software module including components arranged in a first order, a first one of the components including a reference to a location in a second one of the components, the second one of the components preceding the first one of the components in the first order, and arranging the components into a second order." As discussed above, Levine fails to teach or describe a software module having a reference that refers to a location within that

software module. Instead, Levine describes the tsort program's use of the output of the lorder program as being a dependency list to resolve symbols. Symbol resolution is not the equivalent of arranging the components into a second order. Thus, Applicant respectfully submits that for at least the reasons discussed above with respect to claim 1, claim 55 should also be allowed. Therefore, Applicant requests that the rejection of claim 55, and the rejection of claim 56-60, which depend from and include all the limitations of claim 55, be withdrawn.

III. THE 35 U.S.C. § 103(a) REJECTIONS SHOULD BE WITHDRAWN

The Examiner has rejected claim 42 under 35 U.S.C. §103(a) as unpatentable over Levine in view of U.S. Patent No. 6,185,733 to Breslau (hereinafter "Breslau"). (*See* Office Action, pp. 10-11, ¶¶ 9-10) As discussed above, Levine does not teach or suggest all the limitations of independent claims 1. Breslau does not cure the defects of Levine. Accordingly, because claims 42 depends from and, therefore, includes all of the limitations of corresponding independent claim 1, it is respectfully submitted that claim 42 is also allowable over the cited references.

CONCLUSION

In light of the foregoing, Applicant respectfully submits that all of the now pending claims are in condition for allowance. All issues raised by the Examiner having been addressed. An early and favorable action on the merits is earnestly solicited.

Respectfully submitted,

Dated: December 27, 2005

By: _____
Michael J. Marcin (Reg. No. 48,198)

Fay Kaplun & Marcin, LLP
150 Broadway, Suite 702
New York, NY 10038